

# *Scalable Distributed Video-on-Demand: Theoretical Bounds and Practical Algorithms*

Laurent Viennot — Yacine Boufkhad — Fabien Mathieu — Fabien de Montgolfier — Diego  
Perino

**N° 6496**

Avril 2008

Thème COM

 *apport  
de recherche*



# Scalable Distributed Video-on-Demand: Theoretical Bounds and Practical Algorithms

Laurent Viennot\*, Yacine Boufkhad†, Fabien Mathieu‡, Fabien de Montgolfier†, Diego Perino‡

Thème COM — Systèmes communicants  
Projet GANG

Rapport de recherche n° 6496 — Avril 2008 — 19 pages

**Abstract:** We analyze a distributed system where  $n$  nodes called *boxes* store a large set of videos and collaborate to serve simultaneously  $n$  videos or less. We explore under which conditions such a system can be scalable while serving any sequence of demands. We model this problem through a combination of two algorithms: a video allocation algorithm and a connection scheduling algorithm. The latter plays against an adversary that incrementally proposes video requests.

Our main parameters are: the ratio  $u$  of the average upload bandwidth of a box to the playback rate of a video; the maximum number of connections  $c$  used for downloading a video; the number  $m$  of distinct videos stored in the system, i.e. its catalog size. In an homogeneous system (i.e. all node capacities are equal) where a box downloads its video with no more than  $c$  equal rate connections, we give necessary conditions for achieving scalable catalog size. In particular, we prove for that case a lower bound  $u \geq \max\{1 + \frac{1}{c}, \mu\}$ , where  $\mu \geq 1$  is the maximum growth factor of any swarm of boxes viewing the same video during a period of time equivalent to start-up delay (our model tolerates swarms growing exponentially with time). On the other hand, we prove that catalog size  $\Omega(n)$  can be achieved with a centralized scheduling algorithm when  $u \geq \max\{1 + \frac{1}{c}, \mu\}$ ,  $c \geq 2$  and nodes are reliable.

Additionally, we propose a distributed connection scheduling algorithm associated to a random video allocation scheme for heterogeneous systems where box upload capacity is proportional to storage capacity. It achieves catalog size  $\Omega(n/\log n)$  and allows to successfully handle a sequence of  $O(n)$  adversarial events with high probability as long as  $u \geq \mu + \frac{1}{c}$ . As a special case, it can be used to solve single video distribution with  $O(1)$  reliable seed boxes, or  $O(\log n)$  unreliable seed boxes, with constant capacities.

**Key-words:** video-on-demand, scalability, peer-to-peer

Supported by ANR project “ALADDIN”.

Supported by collaborative project “MARDI II” between INRIA and Orange Labs.

\* INRIA Rocquencourt, France

† LIAFA, Paris, France

‡ Orange Labs, Issy-les-Moulineaux, France

# Passage à l'échelle de services distribués de vidéos-à-la-demande

**Résumé :** Nous considérons un système de  $n$  nœuds (*boîtes*) qui hébergent un ensemble de films et cherche à diffuser jusqu'à  $n$  flux vidéos simultanés. Une question qui se pose est de savoir sous quelles conditions un tel système peut passer à l'échelle tout en supportant n'importe quelle séquence de demandes. Ce problème se décompose en deux parties: la répartition initiale des vidéos dans les boîtes et l'allocation des ressources en fonction des demandes. Pour ce dernier problème, nous supposons qu'un adversaire émet des demandes de manière incrémentale.

Les principaux paramètres du problème sont : le rapport  $u$  entre l'upload moyen des boîtes et le débit nécessaire à la lecture de la vidéo ; le nombre maximal  $c$  de connections utilisables dans la récupération d'une vidéo ; le nombre  $m$  de films distincts stockés dans le system (la taille du catalogue).

Dans un système homogène (toutes les boîtes ont les mêmes capacités), nous donnons à  $c$  fixé les conditions nécessaires à la réalisation d'un système capable de passer à l'échelle. Nous montrons en particulier que  $\max\{1 + \frac{1}{c}, \mu\}$  est une borne inférieure pour  $u$ ,  $\mu \geq 1$  étant le facteur de croissance maximal des ensembles de demandes d'une vidéo pendant une période de temps de l'ordre du temps d'amorce de lecture d'une vidéo (notre modèle tolère ainsi une croissance exponentielle des demandes).

Réciproquement, nous prouvons que si  $u \geq \max\{1 + \frac{1}{c}, \mu\}$ ,  $c \geq 2$  et si les boîtes sont fiables, alors il est possible d'avoir une taille de catalogue  $m = \Omega(n)$ , avec un algorithme d'allocation centralisé.

Enfin, nous proposons un algorithme d'allocation distribué, associé à un algorithme de répartition aléatoire adapté aux systèmes hétérogènes où la capacité d'upload des boîtes est proportionnelle à leur capacité de stockage. Il est alors possible de servir une séquence de  $O(n)$  demandes adversariales avec forte probabilité, avec une taille de catalogue en  $\Omega(n/\log n)$ , à la condition d'avoir  $u \geq \mu + \frac{1}{c}$ .

**Mots-clés :** vidéo-à-la-demande, passage à l'échelle, pair-à-pair

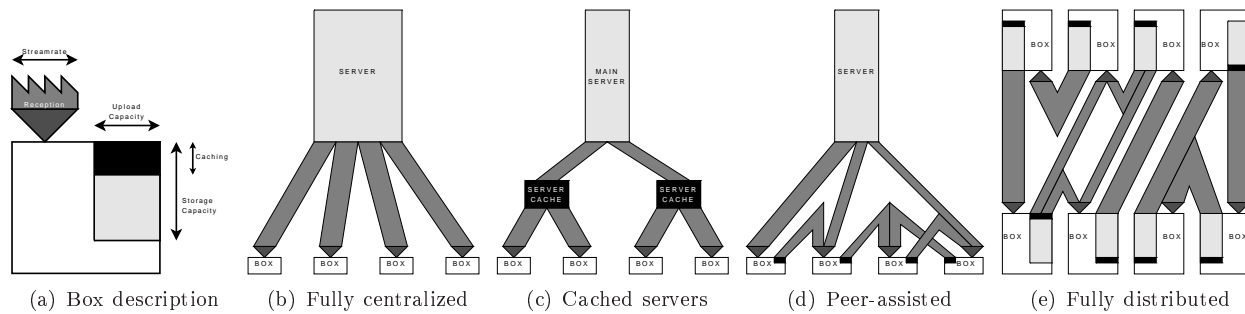


Figure 1: Generic box description, and possible Video-on-Demand architectures

## 1 Introduction

### 1.1 Background

The quest for scalability has yielded a tremendous amount of work in the field of distributed systems in the last decade. Most recently, the peer-to-peer community has grown up on the extreme model where small capacity entities collaborate to form a system whose overall capacity grows proportionally to its size. Historically, first peer-to-peer systems were devoted to collaborative storage (see, e.g., [11, 22, 13]). The academic community has proposed numerous distributed solutions to index the contents stored in a such a system. Most prominently, one can mention the numerous distributed hash table proposals (see, e.g., [21, 23, 20, 24]). Extreme attention has then been paid to *content distribution*. There now exists efficient schemes for single file distribution [8]. Several proposals were made to cooperatively distribute a stream of data (see, e.g., [5, 17, 26, 27, 14, 10]). The main difficulty in streaming is to obtain low delay and balanced forwarding load. Most recently, the problem of collaborative video-on-demand has been addressed. It has mainly been studied under the *single video distribution* problem: how to collaboratively download a video file and view it at the same time [15, 6, 3, 2, 14, 12, 7, 16, 9]. This somehow combines both file sharing and streaming difficulties. On the one hand, participants are interested by different parts of the video. On the other hand, an important design goal resides in achieving a small *start-up delay*, i.e. the delay between the request for the video and the start of playback.

Most of these solutions rely on a central server for providing the primary copy of a video to the set of entities collaboratively viewing it. Following the pioneering idea of Suh et al. [25], we propose to explore the conditions for achieving fully distributed scalable video-on-demand systems. One important goal is then to enable a large *distributed catalog*, i.e. a large number of distinct primary video copies distributively stored. We thus consider the entities storing the primary copies of the videos as part of the video-on-demand system. This model can encompass various architectures like a centralized system with download-only clients, a peer-assisted server as assumed in many proposed solutions, a distributed server with download-only clients or a fully distributed system as proposed in [25]. These scenarios are illustrated by Figure 1. The fully distributed architecture is mainly motivated by the existence of set-top boxes placed directly in user homes by Internet service providers. As these boxes may combine both storage and networking capacities, they become an interesting target for building a low cost distributed video-on-demand system that would be an alternative to more centralized systems.

### 1.2 Related Work

A significant amount of work has been done on *peer-assisted* video-on-demand, where there is still a server (or a server farm) which stores the whole catalog. Annapureddy *et al.* [3] investigate the distribution (on-demand) of a single video. They propose an algorithm that uses a combination of network coding, segment scheduling and overlay management in order to handle high streamrates and slow start-up delays even under *flashcrowds* scenarios. This follows an approach similar to [14] consisting in grouping viewers of the same segment of the video together. Adaptations of the BitTorrent protocol to the single video distribution are proposed in [16, 7]. Cheng & al. propose [6] connections to nodes at different position in the video to enable VCR-like features (seeking, fast-forwarding, ...). A thorough analysis of single video distribution under Poisson arrival is made in [15], strategies for pre-fetching of future content are simulated against real traces. Caching strategies are tested against real traces in [2]. It is proposed in [16] to use a distributed hash table to index videos cached by

each node. However, there is no guarantee that the videos stay in cache. All these solutions rely on a centralized server for feeding the system with primary copies of videos.

To the best of our knowledge, only a few attempts have been made so far to investigate the possibility of a server-free video-on-demand architecture. Suh *et al.* proposed the Push-to-Peer scheme [25] where the primary copies of the catalog are pushed on set-top boxes that are used for video-on-demand. The paper addresses the problem of fully distributing the system (including the storage of primary copies of videos), but scalability of the catalog is not a concern. Indeed, a constant size catalog is achieved: each box stores a portion of each video. A code-based scheme is combined to a window slicing of the videos and a pre-fetching of every video. The paper is mainly dedicated to a complex analysis of queuing models to show how low start-up delay and sufficiently fast download of videos can be achieved. The system is tailored for boxes with upload capacity lower than playback rate. As we will see, this is a reason why scalable catalog cannot be achieved in this setting.

Finally, in a preliminary work [4], we begun to analyze the conditions for catalog scalability. This work mainly focuses on the problem of serving pairwise distinct videos with a distributed system with homogeneous capacities and no node failure. Most notably, an upper bound of  $n + O(1)$  is shown for catalog size when upload is too scarce. A distributed video-on-demand is sketched based on pairwise distinct requests and using any existing single video distribution algorithm for handling multiply requested videos. We extend much further this work to multiple requests, heterogeneous case and node churn scenarios. We can now provide an upper bound of  $o(n)$  for catalog size when upload is scarce and multiple requests are allowed. Secondly, we prove that the maximum flow technique proposed for pairwise distinct requests can be extended to answer any demand with possible multiplicity. This requires a much more involved proof. Additionally, we give insight on heterogeneous systems where nodes may have different capacities one from another. Finally, we propose a distributed algorithm combining both primary video copy distribution and replication of multiply requested videos. Let us now give more details about the contributions of the present paper.

### 1.3 Contribution

This paper mainly proposes a model for studying the conditions that enable scalable video-on-demand. Most importantly, we focus on scalable catalog size and scalable communication schemes. Our approach consists in first formulating necessary requirements for scalability and then try to design algorithms based on these minimal assumptions. We call *boxes* the entities forming the system. Most notably, we require that a box downloads a video using a limited number  $c$  of connections. This is a classical assumption for having a scalable communication maintenance cost in an overlay network. Note that efficient  $n$ -node overlay network proposals usually try to achieve  $c = O(\log n)$ . Equivalently, we assume that video data and video stream cannot be divided into infinitely small units. With at most  $c$  connections, a single connection should have rate at least  $\frac{1}{c}$  where 1 corresponds the normalized playback rate of the video. Similarly, as connections have to remain steady during long period of times with regard to start-up delay  $t_S$ , a box should store portions of video data of size at least  $\frac{c}{t_S}$ . These assumptions of minimal unit of data or minimal connection rate provided by a box of the system are particularly natural when one faces the problem of distributing video data on several entities: one have to define some elementary chunk size and distribute one or more of them per entity.

We first show that these discrete nature assumptions on connection rates and chunk size give raise to an upload bandwidth threshold. If the *average upload*  $u$  is no more than 1, scalable catalog size cannot be achieved, a minimal average upload of  $1 + \frac{1}{c}$  is thus required. Theorem 2 states this as soon as  $c = O(n^\varepsilon)$  for any  $\varepsilon < \frac{1}{2}$  (e.g.,  $c$  is constant or bounded by a poly-logarithmic function of  $n$ ). Moreover, a distributed video-on-demand system cannot achieve scalable catalog size if the number of arrivals for a given video increases too rapidly. We call *swarm* of a video the set of boxes playing it. If the swarm of a video can increase by a multiplicative factor  $\mu > 1$  during a period equivalent to start-up delay  $t_S$ , then it is necessary to have upload  $u \geq \mu$  to replicate sufficiently quickly the video data (see Theorem 1). These lower bounds on  $u$  mainly rely on the assumption that with large catalog size, some video must be replicated on a limited number of boxes. (This assumption may be deduced from our bound  $c$  on the number of connections or may be taken for itself).

On the other hand, we give algorithms for enabling scalable video-on-demand. We model the algorithmic part of a video-on-demand system with two algorithms: a *video allocation* algorithm is responsible for placing video data on boxes, and a *scheduling algorithm* is responsible for managing video requests proposed by an adversary, i.e. propose connections for each box to download its desired video. We build two scheduling algorithms based on random allocation of video data. Let us first remark that is not possible to resist node failures if some video has its data on a limited number of boxes: an adversary can place node failure events on these boxes and then request the video. We thus propose a first scheduler under the assumption that no node fails and that we meet the conditions  $u \geq \max\{1 + \frac{1}{c}, \mu\}$  and  $c \geq 2$ . The problem of finding suitable connections for

$n$	Number of boxes for serving videos.
$m$	Number of videos stored in the system (catalog size).
$d_i$	Storage capacity of box $i$ (in number of videos).
$d$	Average storage capacity of boxes.
$k$	Number of duplicates copies of a video with random allocation ( $k \approx nd/m$ )
$u_i$	Upload capacity of box $i$ (in number of full video streams).
$u$	Average upload capacity of boxes.
$c$	Maximum number of connections for downloading a video.
$s$	Number of stripes of videos (a video can be viewed by downloading its $s$ stripes simultaneously).
$a$	Minimum ratio of active boxes in an homogeneous system.
$t_S$	start-up delay: maximum delay to start playing a video.
$v_S$	Maximum number of arrivals during $t_S$ for a video not being played.
$\mu$	Bound on swarm growth: if a swarm has size $p$ at time $t$ , it has size less than $\mu p$ at time $t + t_S$ .

Table 1: Key parameters

downloading all videos reduce to a maximum flow problem for a given set of requests and a given allocation of videos. We thus propose a centralized scheduler running a maximum flow algorithm. If a centralized tracker for orchestrating connections has already been proposed in several peer-to-peer architectures, it is not clear whether this maximum flow computation could be made in a scalable way. The benefit of this algorithm is thus mainly theoretical. It allows to understand the nature of the problem. Theorem 3 states that a random allocation enables a catalog of size  $\Omega(n)$  and allows to manage any infinite sequence of adversarial requests with high probability (as long as the adversary cannot propose node failures). The problem of scalable video-on-demand can thus be solved with optimal upload capacity in theory. Interestingly, this scheme allows to show that the best catalog size is obtained when the storage capacity of boxes is proportional to their upload capacity.

Additionally, we propose a randomized distributed scheduler based on priority to *playback caching*, i.e. relying on the fact that boxes playing a video can redistribute it. Giving priority to such connections allows to be resilient to exponential swarm growth. We show that with the random allocation of  $\Omega(n/\log n)$  videos in a system where average storage capacity is  $d = \Omega(\log n/c)$  per box, this scheduler can manage  $O(n)$  realistic adversarial events with high probability under the assumption that  $u \geq \mu + \frac{1}{c}$  and the adversary is not aware of the scheduler and allocation algorithm choices (see Theorem 4). Interestingly, our use of playback caching allows to build disjoint forwarding trees for video data in a way similar to Splitstream [5]. The main difference is that relaying nodes buffer data before forwarding it and tree levels are ordered according to the playing position in the video.

The paper is organized as follows. Section 3 exposes the requirements that are needed for the catalog to be scalable. Section 4 investigates the worst case analysis of the problem with no failures; while Section 5 considers more realistic conditions. Then Section 6 proposes to confirm the results of previous sections by the dint of simulations. Some proofs are in given in appendix due to space limitations. We now introduce our model for video-on-demand systems and the notations used throughout this paper.

## 2 Model

We first introduce the key concepts of video-on-demand systems and discuss the associated parameters. We first describe the nodes (often called boxes) of the system, then detail how they may connect to each other to exchange data. We then explain how we decompose the algorithmic part of the system and describe adversary models for testing our algorithms.

**Video system.** We consider a set of  $n$  boxes used to serve videos among themselves. Box  $i$  has storage capacity of  $d_i$  videos and upload capacity equivalent to  $u_i$  video streams. For instance if  $u_i = 1$ , box  $i$  can upload exactly one stream (we suppose all videos are encoded at the same bitrate, normalized at 1). Such a system will be called an  $(n, u, d)$ -video system where  $u = \frac{1}{n} \sum_{i=1}^n u_i$  is the average upload capacity and  $d = \frac{1}{n} \sum_{i=1}^n d_i$  is the average storage capacity. A system is *homogeneous* when  $u_i = u$  and  $d_i = d$  for all  $i$ . Otherwise, we say it is *heterogeneous*. The special case when storage capacities are proportional to upload capacities (i.e.  $d_i = \frac{d}{u} u_i$  for all  $i$ ) is called *proportionally heterogeneous*.

The box activity is defined as a *state*. Box  $i$  is *active* when it can achieve a stable upload capacity no less than  $u_i$  or *inactive* otherwise (e.g. when it is under failure or turned off by user). We suppose that the ratio  $a$  of active boxes remains roughly constant. We assume that the nodes with higher capacity are not more prone to failure than the other nodes, so the average upload capacity of active boxes remains larger than  $u$ . An active box may be *playing* when it downloads a video or *idle* otherwise. The set of boxes playing the same video  $v$  is called *swarm*. Node churn occurs as sequence of *events* consisting in changing the state of a box. *Swarm churn* designates the events concerning a given swarm. We will see in Section 3.1 that scalability cannot be achieved when a swarm grows too rapidly. We thus assume a bounded *growth* factor  $\mu$ : during a period of time  $t_S$  ( $t_S$  is defined below), the size of a swarm is multiplied by a factor  $\mu$  at most. More precisely, and to remove any quantification issues, we assume that the number of events for a given swarm and a given period of time  $t$  is at most  $\mu^{t/t_S}$ . (For convenience, we aggregate the various types of swarm churn within the same bound).

**Connections.** We assume that finding, establishing and setting up a small buffer for starting video playback takes time. We call *start-up delay* the maximal duration  $t_S$  for a box to connect to other boxes and begin playback. We consider that the number  $c_n$  of connections for downloading a video is bounded by some constant  $c$ . The reason is that with constant swarm churn rate, a box will have to change  $\Omega(c_n)$  connections per unit of time. As changing a connection has some latency  $\Omega(t_S)$ , this number should remain bounded or grow very slowly with  $n$ . In connection with this assumption, we suppose that the data of video cannot be split in infinitely small pieces. We thus consider that a connection has minimal rate  $\frac{1}{c}$  (this is obviously the case when connections rates are equally balanced, and it can be modeled by aggregating unitary connections otherwise). Therefore the minimal piece of video data stored on a box is  $\Omega(\frac{1}{c})$  (a trivial lower bound of  $\frac{t_S}{c}$  follows from previous assumptions).

A peer-to-peer video system without any external video sourcing relies on the possibility to replicate a video as it becomes more popular and the number of requests for the video increases. The most straightforward way to do this is to cache in each box the video it is currently playing, which is natural if we want to provide some VCR functionalities. We call *Playback caching* this facility: boxes of a swarm can serve as a relay for the boxes viewing a former part of the video. Note, that in order to bring some flexibility in the swarm, the video can be split into time windows, thus allowing to avoid linear viewing. Time windowing also allows to reduce the problem to the case where all videos have approximately the same duration.

**Video data manipulations.** We consider that all videos have same playback rate, same size and same duration (all three equal to 1 as they are taken as reference for expressing quantities). To enable multi-source upload of a video, each video may be divided in  $s$  equal size *stripes* using some balanced encoding scheme. The video can then be viewed by downloading simultaneously the  $s$  stripes at rate  $1/s$ . A very simple way of achieving stripping consists in splitting the video file in a sequence of small packets. Stripe  $i$  is then made of the packets with number equal to  $i$  modulo  $s$ . Note that our connection number limitation imposes  $s \leq c$ . There are two main reasons for using stripes: it allows to build internal-node-disjoint trees as discussed in Section 5 and it let a box upload sub-streams of rate  $\frac{1}{s}$  to fully use its upload capacity. Stripes may also enable redundancy through correcting codes at the cost of some upload overhead: downloading  $(1 - \varepsilon)s$  stripes is then sufficient to decode the full video stream (e.g. using LT-codes [18] or rateless encoding [19]). For the sake of simplicity, we assume that  $s$  can be large enough to consider all  $u_i s$  and  $d_i s$  as integrals. As mentioned previously, a video can be distributed among several boxes by splitting it according to time windows. However, considering all the time windows of all videos being played at given time, we are back to the same problem fundamentally. For that reason, we do not develop time windowing.

**Video scheme.** A *video allocation* algorithm is responsible for placing primary copies of each video in the system respecting storage capacity constraints of boxes. The most simple scheme consists in storing them statically: video data may be replicated but primary copies of videos are static. Video allocation only changes when new boxes are added to the system or when the catalog is updated. For instance, re-allocating primary copies under node churn would not be practical when live connections consume most of the upload capacity of the system. We assume that the catalog renewal is made at a much larger time scale. Its size and storage allocation are thus considered fixed during a period of several playback times. We may assume that the catalog remains the same during such periods. Of course as the system evolves over a long period of time, some videos are added or removed. The *catalog size* is the number of distinct videos allocated.

When a box state changes, a *scheduling algorithm* decides how to update the connections of playing boxes so that their video is downloaded at rate greater than 1 and all box upload capacities are respected. For



our theoretical bounds (Section 4), we use a centralized scheduler that has full knowledge of the system. For practical algorithms (Section 5), we consider distributed scheduling algorithms: each time a box changes its state, it runs the scheduling algorithm on its own. The scheduling algorithm succeeds if it can establish connections to download the full video stream in time less than  $t_S$ .

We call *video scheme* a combination of an allocation scheme and a scheduling algorithm. We say that a video scheme *achieves* catalog size  $m$  if the allocation scheme can store  $m$  videos in the system so that the scheduling algorithm succeeds in handling all requests of an adversary. The *adversary* knows the list of videos in the catalog and proposes any sequence of node state changes that respects our model assumptions. In its weaker form, it is not aware of the decisions made by the allocation and scheduling algorithms. This is a realistic assumption as there is no reason for user requests to be correlated to something the users of the system are not aware of. Worst case analysis is obtained with the *strong adversary* which is the most powerful adversary possible. It is additionally aware of the choices made by the allocation and scheduling algorithms. In particular, it knows which boxes contain replicas of a given video. If not specified, the adversary is not strong.

### 3 Necessary Conditions for Catalog Scalability

Let us first give some trivial requirements. The total upload is at most  $un$  and, as all active boxes may be playing, the total download capacity needed may be  $n$  so we trivially deduce the lower bound  $u \geq 1$ . As the total storage space of active boxes can be as low as  $adn$  (assuming that average storage capacity of active boxes remains  $d$ ), we have  $m \leq adn$ .

Let us first remark that if we release the constraint on bounded connectivity, then ideal storage of  $adn$  videos can theoretically be achieved in any proportionally heterogeneous  $(n, 1, d)$ -video system when  $c = n$ . As stated in the homogeneous case [25], *full stripping* can achieve this. It consists in splitting each video in  $n$  stripes, one per box. Viewing a video then requires to connect to all other boxes. This result can easily be generalized to the proportionally heterogeneous case with node failures using correcting codes. Such schemes are unpractical for large  $n$  but give a theoretical solution.

On the other hand, we show that some upload provisioning is necessary in our more realistic model. The main hypothesis implying these results is that some video is replicated on  $O(\frac{n}{m})$  boxes at most, i.e.  $o(n)$  if catalog scales. First note that as soon as a video spans at most  $o(n)$  boxes, the system cannot tolerate  $n$  strong adversarial events. Indeed, the strong adversary can propose failure events on all boxes possessing a given video and then propose a request with the video.

#### 3.1 Maximal Swarm Churn Rate

We now state that arrival rate in a given swarm must be lower than average upload. This is our first non trivial lower bound on average upload.

**Theorem 1** *Any homogeneous  $(n, u, d)$ -video system achieving catalog size  $m$  and resilient to swarm growth  $\mu$  satisfies  $u \geq \max\{2, \mu - O(\frac{1}{m})\}$*

For small start-up delay, a realistic value of  $\mu$  would certainly be less than 2. Scalable catalog size is then achievable for  $u \geq \mu$  only.

**Proof.** We consider a scenario where boxes are viewing different videos, and all of them switch to the same video forming a swarm with growth factor  $\mu$ . The swarm of the video has thus size  $v_S$  at time 0,  $v_S\mu$  at time  $t_S$ ,  $v_S\mu^2$  at time  $2t_S$ , and more generally size  $v_S\mu^i$  at time  $it_S$ . We choose a video that is replicated at most  $k = O(\frac{n}{m})$  times in the system. If this data is possessed by sufficiently many boxes, it can be replicated  $k$  times initially. Consider the number of times  $x_i$  the data of the video is replicated outside the swarm at time  $it_S$ . Suppose that all boxes possessing the video either serve new arrivals or pro-actively replicate it with their remaining bandwidth. We then have  $v_S\mu^{i+1} + x_{i+1} \leq v_Su\mu^i + (u-1)x_i$  as the video data must be received by all boxes in the swarm and boxes outside the swarm that replicate it. Suppose  $u \leq \mu$  (otherwise the proof is already over). We get  $x_{i+1} \leq (u-1)x_i$ , and thus  $x_i \leq (u-1)^i k$  as  $x_0 \leq k$ . The former inequality thus gives  $u + \frac{k}{v_S} \left(\frac{u-1}{\mu}\right)^i \geq \mu$ . If  $u < 2$ , we obtain for  $i = \log_\mu n$  that  $u \geq \mu - \frac{k}{v_S n} = \mu - O(\frac{1}{m})$ .  $\square$

Additionally, we can prove that a strict upload of 1 is not sufficient even under low pace arrivals.

### 3.2 Upload Capacity versus Catalog Size

We thus assume in this section that  $c = O(n^\varepsilon)$  for some  $\varepsilon > 0$ . This is for example the case when  $c$  is a poly-log of  $n$  as often assumed in overlay networks [21, 23, 24]. (The rest of the paper assumes a constant  $c$ ). With this bound, we can establish the following trade-off between average upload capacity and achievable catalog size.

**Theorem 2** *For any  $\varepsilon > 0$ , an homogeneous  $(n, u, d)$ -video system with  $u \leq 1$  and  $c = O(n^\varepsilon)$  that can play any demand of  $n$  videos in the no failure strong adversary model has catalog size  $m = O(n^{1/2+\varepsilon})$ .*

The above result states that a video system with scarce capacity poorly scales with  $n$ . As it is valid in the no failure strong adversary model, it remains valid in the strong adversarial model. With our discrete vision of connections, it implies that a minimal upload  $u \geq 1 + \frac{1}{c}$  is necessary for scalability.

**Proof.** Suppose there exists  $\varepsilon > 0$  with  $\varepsilon < \frac{1}{2}$  such that  $c < n^\varepsilon$ . As discussed in Section 2, we use our assumption that a box stores no less than  $\frac{ts}{c}$  data of a given video.

Suppose by contradiction that there exists a video system with catalog size  $m > \frac{2d}{ts}n^{1/2+\varepsilon} > \frac{2dc}{ts}\sqrt{n}$ . As the overall storage capacity is  $dn$ , there exists some video  $v$  whose data is replicated at most  $\frac{dn}{m} \leq \frac{ts}{2c}\sqrt{n}$  times. As useful portion of data of  $v$  have size at least  $\frac{ts}{c}$ , the set  $E$  of boxes storing data of  $v$  has size at most  $\frac{1}{2}\sqrt{n}$ . Let  $F = \overline{E}$  be its complementary. Set  $p = |E| \leq \frac{1}{2}\sqrt{n}$  and  $q = |F|$ .

Now consider the possible request sequence where all boxes  $b_1, \dots, b_q$  of  $F$  successively begin to play  $v$  while boxes of  $E$  play videos not stored at all among boxes in  $E \cup \{b_q\}$ . Box  $b_i$  can download  $v$  from  $E_i = E \cup \{b_1, \dots, b_{i-1}\}$ . Boxes of  $E$  can only download from  $F' = F \setminus \{b_q\}$ .

Suppose that data of  $v$  flows from  $E$  to  $F'$  at rate  $p'$  and from  $E$  to  $b_q$  at rate  $p''$ . We have  $p' + p'' \leq p$  since the overall upload capacity of  $E$  is  $p$ . Data of  $v$  flows internally to  $F'$  at rate at least  $q - 1 - p'$ . The remaining upload capacity to serve  $E$  is thus  $p' - (1 - p'') \leq p - 1$  as  $E$  must additionally serve  $b_q$  at rate  $1 - p''$ . This implies that the number of videos not stored at all on  $E \cup \{b_q\}$  is at most  $p - 1$ . (Otherwise, we have a request that cannot be satisfied.)

As a box contains data of  $\frac{dc}{ts}$  distinct videos at most. We thus deduce  $m \leq \frac{dc}{ts}(p + 1) + p - 1 \leq \frac{dc}{ts}\sqrt{n} < \frac{d}{ts}n^{1/2+\varepsilon}$ . This is a contradiction and we deduce  $m = O(n^{1/2+\varepsilon})$ .  $\square$

We deduce from the previous results that  $u \geq \max\{1 + \frac{1}{c}, \mu\}$  is a minimal requirement for scalability. We now show that it is indeed sufficient.

## 4 Strong Adversary Video Scheme

We now propose a video scheme achieving catalog size  $\Omega(n)$  in the no failure strong adversary model for any video system with average upload  $u \geq \max\{1 + \frac{1}{c}, \mu\}$ . It is based on random allocation of video stripes using  $s = c$  stripes per video and uses a maximum flow scheduler.

### 4.1 Random Allocation

Random allocation consists in storing  $k$  copies of each stripe by choosing  $k$  boxes uniformly at random. This approach was proposed by Boufkhad & al [4] using a purely random graph with independent choices. This has the disadvantage to unbalance the quantity of data stored in each box. We thus prefer to consider a regular bipartite graph where all storage space is used on all boxes. We could obtain the same bounds for the purely random graph. Analysis is slightly more complicated in our case.

For the sake of simplicity, we assume  $k = dn/m$  is an integer. A regular random allocation consists in copying each stripe in  $k$  boxes such that each box contains exactly  $ds$  stripe copies. We model this through a random permutation  $\pi$  of the  $kms$  stripe copies into the  $dns$  storage slots of the  $n$  boxes together: copy  $i$  is stored in slot  $\pi(i)$  (the  $d_1s$  first slots fall into the first box, the  $d_2s$  next slots into the second box, and so on). The best catalog size is obtained for the smallest possible value of  $k$ .

We call *random allocation scheme* the video allocation algorithm consisting in selecting uniformly at random a permutation  $\pi$  and in allocating videos according to  $\pi$ .

### 4.2 Maximum Flow Scheduler

We propose a connection scheduler relying on playback caching. Each time a node state changes, a centralized tracker considers the *multiset of stripe requests*, i.e. the union of all the video stripes being played (some stripes

may be played multiple times) and tries to match stripe requests against boxes so that box  $i$  has degree at most  $u_i s$ . We can model this problem as a flow computation in the following bipartite graph between stripe requests and the boxes storing these stripes. An arc of capacity 1 links every stripe request to all boxes where it is stored (either through the static allocation scheme or through playback caching). The scheduling algorithm consists in running a maximal flow algorithm to find a flow from stripe requests to boxes with the following constraints: each request has an outgoing flow of 1 and such that box  $i$  has incoming flow of  $u_i s$  at most.

We prove that a random regular graph using  $s \leq c$  stripes with  $u \geq \max\{1 + \frac{1}{s}, \mu\}$  has the following property with high probability: for any multiset of  $n$  requests at most, a flow with the desired constraints exists. The proof consists in proving that a random regular allocation graphs has some expander property with high probability. A min-cut max-flow theorem allows to conclude and state the following theorem.

**Theorem 3** *Consider a proportionally heterogeneous  $(n, u, d)$ -video system with  $u \geq \max\{1 + \frac{1}{c}, \mu\}$  and  $c \geq 2$ . Random regular allocation combined with the maximum flow scheduler allows to achieve catalog size  $\Omega(dn/\log_u d)$  and to manage successfully any infinite sequence of strong adversarial events excepting node failures with high probability.*

The proof generalizes in a non trivial manner the proof of [4] that assumes a purely random graph allocation, pairwise distinct requests and homogeneous capacities. Due to space limitations, the proof is given in Appendix A.

### 4.3 Heterogeneous Capacities

As discussed in Appendix A, in the case of heterogeneous capacities, the proof requires the following balance condition. For all set  $E$  of boxes with overall upload capacity  $U_E = \sum_{b \in E} u_b$  and overall download capacity  $D_E = \sum_{b \in E} d_b$  we have for some  $u' \geq \mu + \frac{1}{s}$ :

$$\frac{U_E}{D_E} \geq \frac{u'}{d}$$

(The number of copies per stripe in the allocation graph is then  $k = O(\log_{u'} d)$ ).

Note that  $u' = u$  in the proportionally heterogeneous case and that  $u' \leq u$  in general. Having storage capacity proportional to upload capacity is thus the best situation to optimally benefit from the box capacities.

In the general heterogeneous case, a possible random allocation scheme consists in using only storage  $d'_b = d \frac{u_b}{u''}$  for each box  $b$  for some  $u'' \geq u$  achieving best storage capacity. If box upload capacities are within a constant ratio, this will achieve a catalog size within a constant ratio of the balanced scheme.

### 4.4 Poor Upload Capacity Boxes

Special care has to be taken for an heterogeneous  $(n, u, d)$ -video system where some boxes have upload capacity smaller than  $\mu$ . We say that such boxes are *poor*. The above connection scheduler may be defeated by downloading the same video on a large set  $E$  of such poor boxes, as it may not support exponential growth. This comes from the fact that the storage space for the video coming from playback caching may get larger than  $U_E$ . The above condition on the balance between storage and upload is then violated by playback caching storage.

The general heterogeneous case is reduced to the case where uploads capacities are all greater or equal to  $\mu$  thanks to the following lemma. (This is the last step of the proof of Theorem 3). Due to space limitations, the proof is given in Appendix A.

**Lemma 1** *Consider an  $(n, u, d)$ -video system  $A$  with  $n_P$  boxes of upload less than  $\mu$  having overall upload capacity  $U_P$  and a video allocation scheme with  $s$  stripes satisfying  $u \geq \mu$ . There exists an  $(n, u, d + \frac{n_P - U_P/\mu}{n})$ -video system  $B$  with same video allocation and, for each box  $b$ , upload capacity  $u'_b$  satisfying  $\mu \leq u'_b \leq u_b$ , and same average upload  $u$ , that can emulate any scheme of  $A$  in the no node failure strong adversary model.*

The idea behind this reduction is to statically reserve some upload bandwidth of rich boxes to poor boxes. The average upload of both systems is thus the same. When a poor box  $b$  with upload  $u_b < \mu$  downloads a video, it directly downloads  $u_b s / \mu$  stripes as in the scheduling of  $A$  and downloads the others through relaying by the rich boxes it is associated to. The rich boxes insert also the stripes they forward in their playback cache. This explains why more storage capacity is required. Proof is given in Appendix A.

## 5 Distributed Video Scheme

### 5.1 Purely Random Allocation

The video are stored in the boxes according to a purely random allocation scheme: each stripe of a video is replicated  $k$  times.  $s$  still denotes the number of stripes per video used. Each replica is stored in a box chosen independently at random. Box  $i$  is chosen with probability  $\frac{d_i}{dn}$ . It is possible to add a video in the system as long as the  $k$  chosen boxes have sufficient remaining storage capacity. Such an allocation scheme is qualified as *purely random*.

### 5.2 Playback Cache First Scheduler

We now propose a randomized distributed scheduling algorithm. The main idea of our scheduler is to give priority to playback cache over allocated videos to allow swarm growth  $\mu$ . Only one upload connection is reserved for video allocation uploading. An average upload  $u \geq \mu + \frac{1}{s}$  will thus be required. The scheduling algorithm is split in two parts: stripe searching and connection granting.

*Stripe searching* is the algorithm run by a box for finding another box possessing a given stripe. This algorithm relies on a distributed hash table (or any distributed indexing algorithm) to obtain information about a given stripe. This index allows a box to learn the complete list of boxes possessing the stripe through the video allocation algorithm and a partial list of boxes in the video swarm (i.e. boxes playing the video of the stripe). Stripe searching consists in probing the boxes in these lists until a box accepts a connection for sending the stripe. A connection request includes the stripe requested and the *stripe position* in the stripe file (i.e. an offset position indicating the next octet of video data to be received). A box is eligible for a connection if it has sufficiently many video stripe data ahead that position and if it has sufficiently many upload. This is decided by the connection granting algorithm of the box receiving the request. To give priority to playback-cache forwarding, boxes of the allocation scheme are probed only when the swarm size is less than  $v_S$  or when a stripe is downloaded from a video allocation copy less than  $v_S$  times. To balance upload, several boxes are first probed at the same time, and an accepting box with least number of upload connections for the requested video is selected.

*Connection granting* is the algorithm run by a box that is probed for a connection request. Suppose box  $x$  receives a connection request from box  $y$  for a stripe of video  $v$ . The connection granting algorithm consists in the following steps.

1. If box  $x$  is not viewing  $v$  and is already uploading the stripe, it refuses.
2. If box  $x$  has sufficient upload capacity, it accepts.
3. Otherwise, if box  $x$  is not playing  $v$ , it refuses.
4. Otherwise, if the stripe position of  $x$  for that stripe is not sufficiently ahead the requested stripe position, it refuses.
5. Otherwise, if two or more upload connections of box  $x$  concern a stripe of a video different from  $v$ ,  $x$  selects one of them at random, closes it and accepts box  $y$ .
6. Otherwise, if box  $x$  is uploading the same stripe to some box  $z$  and the requested stripe position of  $y$  is sufficiently ahead the stripe position of  $z$ , it closes the connection to  $z$  and accepts.
7. Otherwise, it refuses.

Note that Steps 4, 5 and 6 can be executed only if box  $x$  plays  $v$ . Step 6 can be executed only if it uploads  $us - 1$  stripes of video  $v$ . (One connection is always reserved to serve allocated stripes). A simple optimization in Steps 6 and 7, consists in *connection flipping*. In Step 6, box  $x$  can send to box  $z$  the address of box  $y$  for re-connecting as the stripe position of  $y$  is sufficiently ahead the stripe position of  $z$  in that case. Box  $z$  can then probe box  $y$  with the same algorithm. In Step 7, box  $y$  can be redirected to any box  $x'$  downloading  $v$  from  $x$  and having stripe position sufficiently ahead the stripe position of  $y$ . Box  $y$  can then probe box  $x'$  with the same algorithm. This way, a box can find its right position according to stripe position in a downloading tree path of its swarm. Similarly, in Step 4, box  $y$  can make a connection flipping with the box from which  $x$  is downloading, and go up the downloading chain until it finds its right position.

Note that this algorithm works in similar manner as Splitstream [5] builds parallel multicast trees for each stripe. The main difference is that each internal node of a tree receives fresh data in a buffer and forwards data which is at least  $t_S$  old. That way, a performance blip within one node will not percolate to all nodes behind it in the sub-tree. Moreover, this ensures that a node has sufficient time to recover from a parent failure. In addition, trees are ordered according to stripe position: boxes with foremost playing position in the video get closer to the root whereas newcomers in the swarm tend to be in lower tree levels. Another interesting point is that nodes downloading from a box with spare number of connections benefit from this free upload capacity and download at a rate faster than needed, allowing to fill their buffer.

### 5.3 Correctness

We cannot prove the resilience of our video scheme against any sequence of adversarial events. The following technical assumption is necessary for our proof and appears as a realistic hypothesis. We assume that a given stripe is searched at most  $O(\log r)$  times on boxes storing it through the video allocation scheme. This requirement is met when the sequence of adversarial events respect the two following conditions. First, a constant number of swarms are started on a given video (a realistic assumption if we consider a period of few playback durations). (There is no restriction on swarm size). Second, node failures are randomly chosen and a given box is chosen with probability  $p_f < 1/v_S$ . A sequence of  $r$  requests is said to be *stress-less* if it satisfies these conditions.

**Theorem 4** *Consider a proportionally heterogeneous  $(n, u, d)$ -video system with  $u \geq \mu + \frac{1}{c}$  and  $\frac{dc}{u} = \Omega(\log n)$ . For any bound  $r = O(n)$ , it is possible to allocate  $\Omega(n/\log n)$  videos and successfully manage  $r$  adversarial stress-less events with high probability.*

To prove this theorem, we analyze a simpler *unitary* video system which can be emulated by any proportionally heterogeneous system with same overall capacities. Again, we choose to use  $s = c$  stripes per video and assume  $u \geq \mu + \frac{1}{s}$ . We view each box  $i$  as the union of  $u_i s$  unitary boxes with upload capacity  $1/s$  (one stripe) and storage capacity  $\frac{d_i}{u_i} = \frac{d}{u}$ . This reduction is indeed penalizing. Consider two unitary boxes that are part of the same real box. In the model, stripes stored on one unitary box can not be uploaded by the other whereas the real box could use two uploads slots for any combination of two stripes of any of the unitary boxes. For some parameter  $k$  made explicit later on, a random allocation of  $k$  replicas per stripe is made according to the purely random allocation scheme described previously. This is equivalent to suppose that each replica is stored in a unitary box chosen uniformly at random since the system is proportionally heterogeneous. As each unitary box has a storage capacity of  $\frac{ds}{u}$  stripes, Chernoff's upper bound allows to conclude that purely random allocation of  $\Omega(dsn/u)$  stripe replicas is possible with high probability when  $\frac{ds}{u} = \Omega(\log n)$ . As we will use  $k = O(\log n)$ , this achieves the required catalog size.

Second, we simplify the scheduler to an algorithm where two schedulers compete. One is allocating *cache stripe* requests within a swarm (i.e. the stripe will be downloaded from a playback caching copy), the other is allocating *seed stripe* requests from the video allocation pool (i.e. the stripe will be downloaded from a unitary box possessing it through the random allocation scheme). We consider that both scheduler operate independently. This is a penalty with regard to practical scheduling, where simple heuristics may reduce considerably the number of conflicts, but it simplifies the stochastic analysis of the system. The cache scheduler allocates swarm stripes and has priority: it operates at real box level according to the above algorithm. From the unitary box point of view of the seed scheduler, the cache scheduler disables some unitary boxes. If the unitary box was uploading some allocated stripe, it is canceled and a seed stripe search is triggered. This is where the reservation of one seed stripe per real box is useful in our analysis. A stripe upload connection is canceled when the real box has at least two of them. As the cache scheduler cancels one box at random uniformly, a given seed stripe is searched at most  $O(\log n)$  times with high probability. The seed scheduler scans the list of unitary boxes possessing the stripe until a free one is found.

Note that a video request in the real system triggers at most  $s$  cache stripe requests and/or  $s$  seed stripe requests. A node failure on a box uploading  $us - 1$  cache stripes results in  $us - 1$  cache stripe requests. Each of them may incur a seed request. The worst event is a video zapping which is equivalent to both events at the same time<sup>1</sup>.  $r$  adversarial requests thus result in  $(u + 1)sr$  seed stripe searches at most.

**Claim 1** *All seed stripe searches succeed with probability greater than  $1 - O(\frac{1}{n})$ .*

<sup>1</sup>In the video zapping event from video  $v$  to video  $v'$ , the box can indeed continue to upload the data of  $v$ , but it cannot continue to download more data. In the worst case, the buffered data may be scarce for all boxes downloading from the box.

**Proof.** We take the point of view of the seed scheduler: a unitary box is free if its real box is active, and the cache scheduler is not using it. As the adversary and the cache scheduler operate independently from stripe allocation, we make the analysis as if the random choices used for stripe allocation were discovered as seed requests arrive. In our case, the purely random scheme consists in allocating each replica in a unitary box chosen uniformly at random. We show that for  $k = O(\log n)$ , each replica is considered at most once with high probability.

For instance, consider a seed request for a stripe  $i$ . Its list of allocated replicas is scanned forward. Each stripe replica falling in an occupied box is discarded until a replica falls in a free unitary box. As observed before, the set  $X$  of unitary boxes that are either under failure or playback cache forwarding is chosen independently from the replica position. The set  $Y$  of unitary boxes uploading seeding stripes depends from independent choices for other stripe replicas. The probability  $p$  that a replica of stripe  $i$  falls in one of the  $t = |X \cup Y|$  occupied unitary boxes is  $p = \frac{|X \cup Y|}{usn}$ . Considering that the number of active boxes is  $n_a \geq an$  and that average upload of active boxes remains  $u$  at least, we obtain that the number of failed unitary boxes is at most  $usn - usn_a$ . As the number of current seed connections is  $|Y|$ , the number of cache connections is at most  $sn_a - |Y|$ . We thus have  $|X| \leq u(n - n_a)s + sn_a - |Y|$  and  $|X \cup Y| \leq u(n - n_a)s + sn_a \leq usn(1 - (1 - \frac{1}{u})a)$ . We thus have  $p \leq 1 - (1 - \frac{1}{u})a$ .

As  $r = O(n)$ , the number of stripe requests is at most  $\lambda n$  for some  $\lambda > 0$ . As discussed previously, the reservation of one stripe for seed connections in real boxes ensures that a given seed connection is discarded with probability at most  $\frac{1}{2}$ . A given stripe is thus discarded at most  $\log_2 \lambda n^2$  times by the cache scheduler with probability  $1 - \frac{1}{\lambda n^2}$  at least. Similarly, for a given stripe, the event that a box uploading it with a seed connection fails happens at most  $O(\log n)$  times with high probability at least according to our stress-less events hypothesis. Every stripe is thus discarded at most  $O(\log n)$  times with high probability. There may be up to  $v_S$  seed connections for a given stripe and stress-less events start at most  $\lambda' \log n$  swarms on the video of the stripe for some  $\lambda' > 0$ . This results in  $v_S \lambda' \log n$  stripe searches at most. Finally, we note that with high probability, every stripe is searched at most  $\lambda'' \log n$  times with high probability for some constant  $\lambda'' > 0$ .

The list of replicas of a stripe can thus be seen as a sequence of zeros (when the replica falls in an occupied unitary box) and ones (when the replica is found). A zero occurs with probability less than  $p$  and a one with probability more than  $1 - p$ . We can conclude the proof if the list of ones in all stripe lists is greater than  $\lambda'' \log n$  with high probability. As random choices for each replica are independent, we conclude using Chernoff's upper bound that a sequence of  $k = O(\log n / (1 - p))$  replicas contains the required number of ones with high probability. (Including the parameters of the model, we use  $k = O(\frac{v_S}{a} \frac{u}{u-1} \log n)$ .)  $\square$

Of course, this vision of consuming the list of replicas of a stripe is particular to our proof. In practice, one can loop back to the beginning of the list when the end is reached.

**Claim 2** *All cache stripe searches succeed with probability greater than  $1 - O(\frac{1}{n})$ .*

**Proof.** We assume a choice of  $s$  such that  $u \geq \mu + \frac{1}{s}$ . As in Lemma 1, we suppose that a box  $i$  with poor upload capacity  $u_i < \mu + \frac{1}{s}$  reserves an upload  $\mu + \frac{1}{s} - u_i$  on some richer boxes. (Note that this augments the probability of failure for the node, a problem we do not try analyze here). A rich box forwarding  $i$  stripes to a poor box accepts preferentially connections for these stripes (as for stripes of the video it is playing) up to an upload bandwidth of  $\mu i$ .

First consider the case where a box  $b$  is entering the swarm (i.e. it requests position 0 in the stripe file). The swarm  $Z$  of  $v$  can be decomposed in the set  $X$  of boxes arrived in  $Z$  before time  $t - t_S$  and the set  $Y$  of boxes arrived in  $Z$  later on. We thus have  $Z = X \uplus Y$  and  $b \in Y$ . If  $X = \emptyset$  then  $|Y| \leq v_S$  according to the arrival bound of our model and each seed stripe search succeeds with high probability as discussed above. On the other hand, if  $X \neq \emptyset$ , we have  $|Z| \leq \mu |X|$  according to the exponential bound on swarm churn in our model. The boxes in  $X$  have overall upload capacity  $(u - \frac{1}{s})|X|$  (including the capacity reserved on richer boxes) and serve at most  $|Z| - \frac{1}{s}$  times the video (box  $b$  is still searching for a stripe). As  $u - \frac{1}{s} \geq \mu$ , some connection slot is free for accepting the stripe connection of box  $b$ . It can always be found if  $b$  has the full list of boxes in the swarm. The fraction of boxes with exceed capacity for the video is thus at most  $\frac{\mu}{u - 1/s}$ . Note that a slightly higher value of  $u \geq \mu + \frac{2}{s}$  would result in a constant fraction of nodes with exceeded capacity for their video. This would allow to find one with high probability if the list of random nodes in the swarm has length  $O(\log n)$ .

Now consider the case where a box  $b$  is reconnecting in its swarm due to some zapping or node failure event. We can prove similarly that the connection flipping algorithm allows to find a node in the swarm to connect to. This relies on the hypothesis that the number of reconnecting nodes at position  $t$  in a video increases by a factor  $\mu$  at most during a period of time  $t_S$  as assumed by our model (all types of swarm churn are aggregated

in the bound  $\mu$ ). □

## 6 Simulations

In this section we evaluate the performance of a practical allocation scheme by the dint of simulations. This scheme is similar to the one described in section 5 but it presents two main differences. Firstly, the storage allocation is based on a random regular graph obtained by a permutation  $\pi$  of the *kms* stripes into the *dns* storage slots. This choice is motivated by the more practical aspect of regular random allocation that allows to completely fill-in boxes. Secondly, once the connections are established, they cannot be re-negotiated when a new video-request is performed. The goal is to test the basic functioning of the algorithm to understand where connection renegotiations become necessary.

We assume that every node has a cache of size 1 where it stores all the stripes of the video it is watching. We suppose video requests arrive at a constant rate, and  $t_S = 2$  minutes.

As stated in previous sections, the efficiency of an allocation scheme depends on the requests pattern. In the following, we use five kind of adversarial schedulers to generate video request sequences:

- **Greedy adversarial.** The greedy adversarial scheduler chooses the request for which the system will select a node with minimal remaining upload bandwidth (among the set of nodes that can be selected by a request in the current configuration). This adversary make greedy decisions. It is strong in the sense that it is aware of video allocation and current connections.
- **Random.** The random scheduler selects a video uniformly at random in the catalog.
- **Netflix.**  $m$  videos are randomly selected from the *Netflix Prize* dataset [1] as catalog for our simulated system. Requests are performed following the real popularity distribution observed in the dataset.
- **Netflix2.** The  $m$  most popular videos of the Netflix Prize dataset are selected as catalog for our simulated system. Requests are performed following the real popularity distribution of these  $m$  videos.
- **Zipf.** The scheduler selects videos following a Zipf's law popularity distribution with  $\gamma = 2$ .

The peers that perform a request follow a sequence of random permutations of the  $n$  peers. All our simulations are performed with  $n = 100$  nodes, and the results are averaged over multiple runs.

### 6.1 Impact of the number of copies per video

We study the maximum number of requests the system is able to satisfy as a function of the number  $k$  of copies per video ( $k \approx \frac{nd}{m}$ ). We suppose that nodes may watch more than one video (for instance if multiple playback devices depend on a single box) so the total number of requests can be larger than  $n$ , even if  $n$  is the typical desired target. We set  $s = 15$ ,  $u = 1 + \frac{1}{s}$  and  $d = 32$ . Figure 2 shows that the system is able to satisfy at least one request per node if  $k \geq 6$ , independently from the requests pattern. Moreover, for the Random, Netflix and Netflix2 schedulers,  $k \geq 3$  is enough.

We indicate as reference the maximum number of requests the system can satisfy considering the global available upload bandwidth. Note, that for  $k \geq 10$ , nodes almost fully utilize their upload bandwidth and the system asymptotically attains the maximum possible number of requests.

### 6.2 Varying the number of stripes

We study the impact of the number of stripes into which videos are split. For this purpose, we set  $k = 10$ ,  $d = 32$  and  $u = 1 + \frac{1}{s}$ .

Figure 3 shows that the system can satisfy  $n$  requests or more for all schedulers but the adversarial. With few stripes, the greedy scheduler may find blocking situations where re-configuration of connections would indeed be necessary. For low  $s$ , more requests are served with other schedulers. This is not surprising, considering that a reduction of the number of stripes leads to an increase of the system global bandwidth. As  $s$  increases,  $u$  tends toward 1 and the number of satisfied requests to  $n$ .

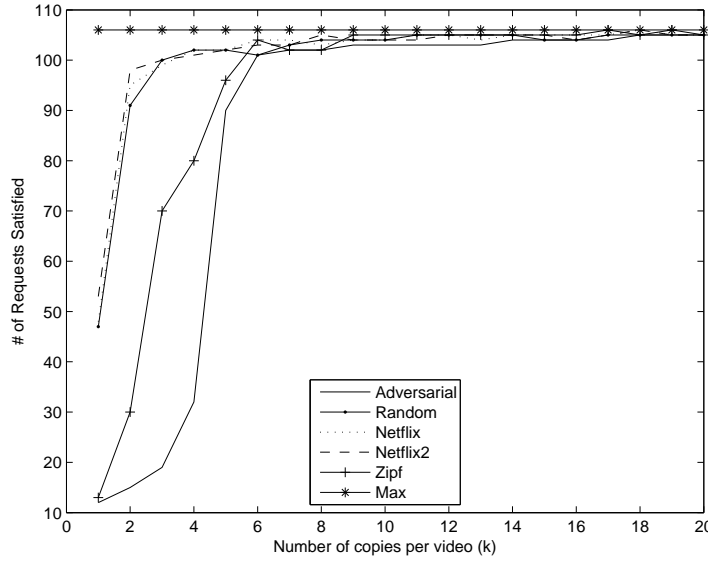


Figure 2: Requests satisfied as a function of  $k$ .  $n = 100$ ,  $d = 32$ ,  $s = 15$ ,  $u = 1 + \frac{1}{s}$

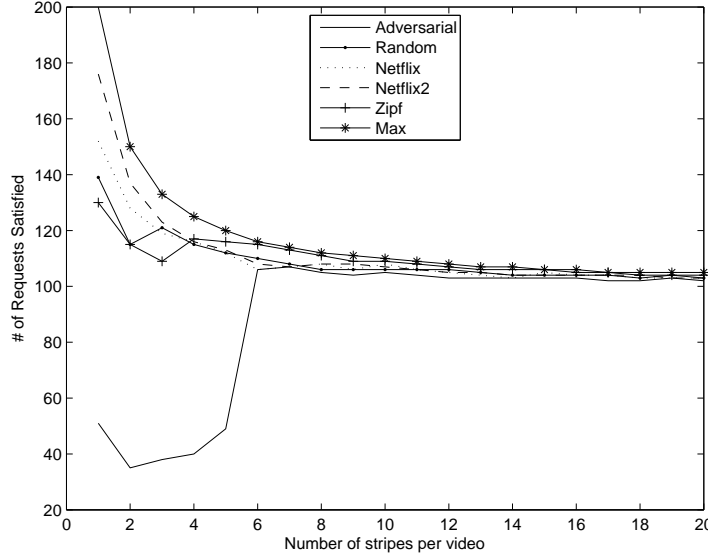


Figure 3: Requests satisfied as a function of  $s$ .  $n = 100$ ,  $d = 32$ ,  $k = 10$ ,  $u = 1 + \frac{1}{s}$

### 6.3 Heterogeneous capacities

We analyze the impact on the number of video requests satisfied in presence of nodes with different upload capacities. Node capacity distribution is a bounded Gaussian distribution with  $u = 1 + \frac{1}{s}$  and different variance values. We set  $k = 10$ ,  $s = 15$  and  $d = 32$ . Figure 4 shows the results. Schedulers can satisfy at least  $n$  requests for small or large values of upload variance, with a slight loss of efficiency between. This may come from the fact that we do not use a proportional allocation scheme here.

### 6.4 Node failures

We evaluate the impact of off-line peers on the number of video requests the system can satisfy. We set  $k = 10$ ,  $s = 15$ ,  $d = 32$  and  $u = 1 + \frac{1}{s}$ . We then randomly select some nodes and we set them inactive for the simulation.

Figure 5 shows the system can satisfy video requests for at least all the active nodes in the system up to 40% failures ( $a = 0.6$ ). Then, a drastic decrease in the performance occurs. As soon as there are 10% of boxes off-line, the adversarial scheduler is able to block the system.



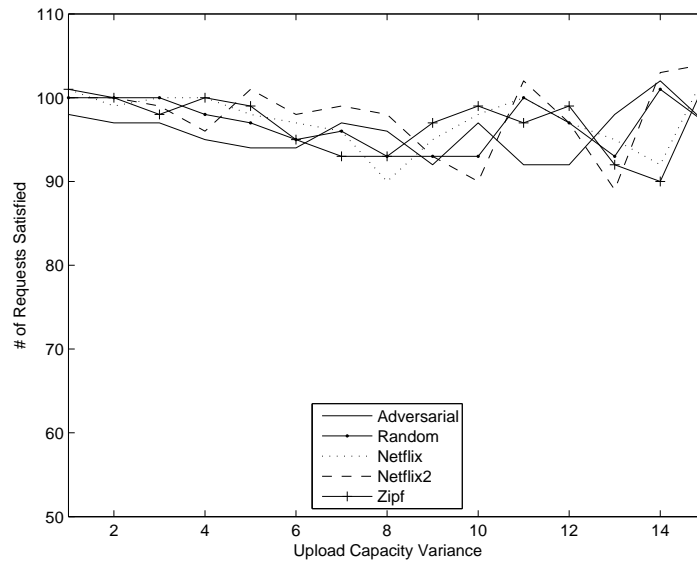


Figure 4: Requests satisfied with heterogeneous capacities.  $n = 100$ ,  $d = 32$ ,  $k = 10$ ,  $s = 15$ ,  $\bar{u} = 1 + \frac{1}{s}$

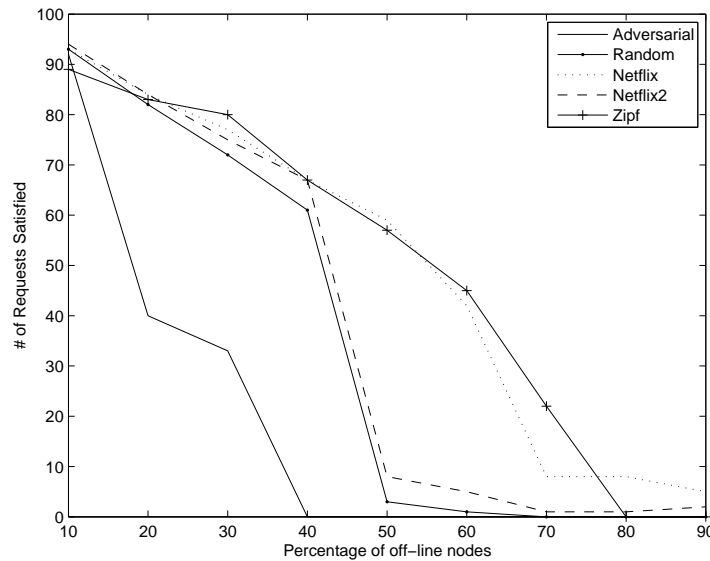


Figure 5: Number of requests satisfied with static off-line peers.  $n = 100$ ,  $d = 32$ ,  $k = 10$ ,  $s = 15$ ,  $u = 1 + \frac{1}{s}$

## 7 Conclusion

In this paper, we show an average upload bandwidth threshold for enabling a scalable fully distributed video-on-demand system. Under that threshold, scalable catalog cannot be achieved. Above the threshold, linear catalog size is then possible and the problem of connecting nodes to serve demands reduces to a maximum flow problem. A slight upload provisioning allows to build distributed algorithms achieving scalability.

## References

- [1] The Netflix prize. <http://www.netflixprize.com/>.
- [2] Matthew S. Allen, Ben Y. Zhao, and Rich Wolski. Deploying video-on-demand services on cable networks. In *Proc. of the 27th Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 63–71, Washington, DC, USA, 2007. IEEE Computer Society.

- [3] Siddhartha Annapureddy, Saikat Guha, Christos Gkantsidis, Dinan Gunawardena, and Pablo Rodriguez. Exploring VoD in P2P swarming systems. In *INFOCOM*, pages 2571–2575, 2007.
- [4] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. Perino, and L. Viennot. Achievable catalog size in peer-to-peer video-on-demand systems. In *Proc. of the 7th Int. Workshop on Peer-to-Peer Systems (IPTPS)*, pages 1–6, 2008.
- [5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP)*, 2003.
- [6] Bin Cheng, Xuezheng Liu, Zheng Zhang, and Hai Jin. A measurement study of a Peer-to-Peer Video-on-Demand system. In *Sixth International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 1–6, 2007.
- [7] Yung Ryn Choe, Derek L. Schuff, Jagadeesh M. Dyaberi, and Vijay S. Pai. Improving VoD server efficiency with BitTorrent. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 117–126, New York, NY, USA, 2007. ACM.
- [8] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.
- [9] Tai Do, Kien A. Hua, and Mounir Tantaoui. P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. In *Proc. of the IEEE Int. Conf. on Communications (ICC 2004)*, jun 2004.
- [10] A.T. Gai and L. Viennot. Incentive, resilience and load balancing in multicasting through clustered de bruijn overlay network (prefixstream). In *Proceedings of the 14th IEEE International Conference on Networks (ICON)*, volume 2, pages 1–6. IEEE Computer Society, September 2006.
- [11] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *Computer Communication Review*, 32(1):82, 2002.
- [12] Yang Guo, Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley. P2cast: peer-to-peer patching scheme for vod service. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 301–309, New York, NY, USA, 2003. ACM.
- [13] S. B. Handurukande, A.-M. Kermarrec, F. Le Fessant, L. Massoulié, and S. Patarin. Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems. *SIGOPS Oper. Syst. Rev.*, 40(4):359–371, 2006.
- [14] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. Insights into pplive: A measurement study of a large-scale p2p iptv system. In *In Proc. of IPTV Workshop, International World Wide Web Conference*, 2006.
- [15] Cheng Huang, Jin Li, and Keith W. Ross. Can internet video-on-demand be profitable? *SIGCOMM Comput. Commun. Rev.*, 37(4):133–144, 2007.
- [16] Vaishnav Janardhan and Henning Schulzrinne. Peer assisted VoD for set-top box based IP network. In *Peer-to-Peer Streaming and IP-TV Workshop (P2P-TV)*, pages 1–5, 2007.
- [17] D. Kotic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh, 2003.
- [18] Michael Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–282, 2002.
- [19] P. Maymounkov and D. Mazières. Rateless codes and big downloads, 2003.
- [20] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.
- [21] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, New York, NY, USA, 2001. ACM.

- [22] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proc. of the 1st IEEE Int. Conf. on Peer-to-Peer (P2P 2001)*. IEEE Computer Society, 2001.
- [23] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.
- [24] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.
- [25] Kyoungwon Suh, Christophe Diot, James F. Kurose, Laurent Massoulié, Christoph Neumann, Don Towsley, and Matteo Varvello. Push-to-Peer Video-on-Demand system: design and evaluation. *IEEE Journal on Selected Areas in Communications, special issue on Advances in Peer-to-Peer Streaming Systems*, 25(9):1706–1716, December 2007.
- [26] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming, 2003.
- [27] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. In *Proc. of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 363–371, 2002.

## Appendix A

### Maximum flow scheduler

We prove Theorem 3 thanks to the two following lemmas. For the sake of clarity, the proof is written for the homogeneous case. It is discussed later on how it generalizes to heterogeneous capacities.

**Lemma 2 (Min-cut max-flow)** *Consider a bipartite graph from  $U$  to  $V$  and an integer  $b > 0$ . There exist a  $b$ -matching where each node of  $U$  has degree 1 and each node of  $V$  has degree at most  $b$  iff each subset  $U' \subseteq U$  has at least  $|U'|/b$  neighbors in  $V$  (i.e., the graph is a  $1/b$ -expander).*

**Proof.** The  $1/b$ -expander property is clearly necessary. We prove it is sufficient by considering the flow network obtained by adding a source node  $a$  and a sink node  $z$  to the bipartite graph. An edge with capacity 1 is added from  $a$  to each node in  $U$ . Edges of the bipartite graph are directed from  $U$  to  $V$  and have capacity 1. An edge with capacity  $b$  is added from each node in  $V$  to  $z$ . The  $1/b$ -expander property implies that every cut has capacity  $|U|$  at least. The well-known min-cut max-flow theorem allows to conclude.  $\square$

**Lemma 3** *Consider a random regular permutation graph of  $kms = dns$  stripe copies into the  $dns$  memory slots of  $n$  boxes. The probability that  $ki$  given copies fall into  $p$  given boxes with  $pds \geq ki$  is less than  $(\frac{p}{n})^{ki}$ .*

**Proof.** Drawing uniformly at random a permutation of the  $kms = dns$  stripes amounts to choose uniformly at random a slot for the first stripe, then a slot for the second among the remaining slots and so on. The  $ki$  stripes are ordered. Let  $E_a$  denotes the event that the  $a^{th}$  copy of stripe falls into one of the  $pds$  slots of the  $p$  boxes.  $P(\cap_{a \leq ki} E_a) = P(E_1).P(E_2|E_1)...P(E_a|E_1 \cap E_2 \dots \cap E_{a-1})... = \frac{pds}{nds} \cdot \frac{pds-1}{nds-1} \dots \frac{pds-a+1}{nds-a+1} \dots \leq (\frac{p}{n})^{ki}$  (since  $\frac{pds-i}{nds-i} \leq \frac{pds}{nds}$  for  $p \leq n$ ).  $\square$

**Proof.[of Theorem 3]** We assume that  $s \leq c$  is sufficiently large to ensure  $u \geq 1 + \frac{1}{s}$ . We suppose  $u \geq \mu$  and  $s \geq 2$ . Consider the multiset of stripe requests at some time  $t$ . Its size is  $ns$  at most as there are no more than  $n$  videos played. Let  $S$  be a sub-multiset of size  $i$  among the requested stripes. Let  $i_1$  be the number of pairwise distinct requests in  $S$  and  $i_2 = i - i_1$  be the number of duplicated requests in  $S$ . As swarm growth is bounded by  $\mu$ , there are at least  $\alpha i_2$  nodes where duplicate request can be downloaded with  $\alpha = \frac{1}{\mu}$ .

Let  $B(S)$  denote the set of boxes from which any stripe of  $S$  may be downloaded. From Lemma 2, a connection matching for serving the request can always be found if no multiset  $S$  of at most  $rs$  requested stripes verifies  $|B(S)| < j$  with  $j = \frac{i}{us}$ . Note that  $B(S)$  includes at least the given boxes where duplicate requests may be downloaded thanks to playback caching. This represents at least  $\alpha i_2$  boxes and  $|B(S)| \geq j$  for  $\alpha i_2 \geq j$ . We may thus consider only  $\alpha i_2 < i/us$  (implying  $i_1 > (1 - 1/\alpha us)i$ ). By summing over all sets of  $j = i/us$  boxes and using Lemma 3, we get the following bound relying only on the stripe copies placed according to the video allocation graph (this probability is 0 for  $i \leq us$ ):  $P(|B(S)| < j) \leq \binom{n}{j} (\frac{j}{n})^{ki_1} \leq (\frac{unse}{i})^{i/us} (\frac{i}{uns})^{ki_1}$ . The last inequality is obtained by using the standard upper bound of the binomial coefficient  $\binom{b}{a} \leq (\frac{be}{a})^a$ .

Using Markov inequality, the probability that some obstruction multiset  $S$  for some request exists is bounded by the expected number of such obstructions. By summing the above inequality over all multisets  $S$  of at most  $ns$  stripes, we get the following bound on the probability  $p$  that the graph cannot satisfy all possible requests:  $p \leq \sum_{i=us}^{ns} \sum_{i_1=i(1-1/\alpha us)}^i M(i, i_1) (\frac{unse}{i})^{i/us} (\frac{i}{uns})^{ki_1}$  where  $M(i, i_1)$  is the number of multisets of cardinality  $i$  taken from sets of stripes of cardinality  $i_1$ .  $M(i, i_1)$  is at most  $M(i, i_1) \leq \binom{\lfloor nds/k \rfloor}{i_1} \binom{i+i_1-1}{i_1-1} \leq (\frac{ndse}{ki})^i \binom{2i}{i} \leq (\frac{4ndse}{i})^i$  since  $i_1 \leq i$  and considering that  $k \geq 1$ . Notice also that  $(\frac{i}{uns})^{ki_1} \leq (\frac{i}{uns})^{ki-ki/\alpha us}$ . The probability is then at most:  $p \leq \sum_{i=us}^{ns} \frac{i}{\alpha us} (\frac{i}{uns})^{\kappa i} \delta^i \leq \frac{n}{\alpha u} \sum_{i=us}^{ns} (\frac{i}{uns})^{\kappa i} \delta^i$  where  $\delta = 4de^{1+1/us}/u$  and  $\kappa = k - k/\alpha us - 1/us - 1$ .

It is easy to check that as a function of  $i$  the terms of the sum  $\phi(i) = (\frac{i}{uns})^{\kappa} \delta^i$  decrease from  $\phi(us)$ , reach a minimum at  $\phi(i^*) = \phi(\frac{uns}{\delta^{1/\kappa e}})$  then increase to  $\phi(ns)$ . Using this fact, we bound  $p$  by considering separately the sum for  $i < i^*$  and  $i > i^*$  and by replacing each term with the maximum term on its side. On one hand,  $\frac{n}{\alpha u} \sum_{i=us}^{\lfloor i^* \rfloor} (\frac{i}{uns})^{\kappa} \delta^i \leq \frac{n}{\alpha u} \cdot ns \cdot \phi(us) = \frac{n}{\alpha u} \cdot ns \cdot \frac{1}{n^{\kappa us}} \delta^{us} \leq O(\frac{1}{n^{\kappa us}})$ . On the other hand, the sum of the terms of rank greater than  $i^*$  gives  $\frac{n}{\alpha u} \sum_{\lfloor i^* \rfloor + 1}^{ns} (\frac{i}{uns})^{-\kappa i} \delta^i \leq \frac{n}{\alpha u} \cdot ns \cdot u^{-\kappa ns} \delta^{ns} \leq O(n^2 (u^{-\kappa} \delta)^{ns})$ . Finally,  $p \leq O(\frac{1}{n^{\kappa us}}) + O((u^{-\kappa} \delta)^{ns})$ . For the first term to vanish, we need  $u^{-\kappa} \delta < 1$  and then  $\kappa > \log_u(\delta)$ . For this, we need to replicate each stripe at least  $k$  is then  $k > \log_u(d) \frac{\alpha us}{\alpha us - 1} + \frac{\alpha + \alpha us \log_u(4e^2)}{\alpha us - 1}$ . For the sake of simplicity,

consider  $s \geq 2$  the lower bound on the number of replicates is  $k > 2 \log_u(d) + 2 \log_u(4e^2) + 1$ . In this case the probability of failure is at most  $p \leq O\left(\frac{1}{n^{\kappa u s}}\right)$  (note that  $\kappa u s > 0$ ) and then the bipartite graph can satisfy all possible requests with high probability. Since the number of videos that can be stored is  $nd/k$  and given the condition on  $k$ , the storage capacity is  $\Omega(nd/\log_u(d))$ .  $\square$

Now consider the heterogeneous case. Lemma 2 and the above proof may be generalized. Recall that box  $b$  has storage capacity  $d_b$  and upload capacity  $u_b$ . The condition for an obstruction then becomes  $\sum_{b \in B(S)} s u_b < |S| = i$ . We can then consider any subset  $E$  of boxes with overall capacities  $U_E = \sum_{b \in E} u_b$  and  $D_E = \sum_{b \in E} d_b$  such that  $U_E < i/s$ . As boxes are chosen according to their capacity, the probability to put a stripe in  $E$  with random allocation is thus  $\frac{D_E}{nd} = \frac{D_E U_E}{nd U_E} < \frac{D_E}{d U_E} \frac{i}{ns}$  for an obstruction. Assuming  $d \frac{U_E}{D_E} \geq u'$  for all  $E$ , we can follow the same tracks for the proof with the probability of obstruction being less than  $\frac{j}{i}$  with  $j = \frac{i}{u' s}$ . With a smallest upload capacity of 1 stripe, the total number of such sets  $E$  is bounded by  $\binom{usn}{i}$  instead of  $\binom{n}{j}$  in the above proof. This larger factor in the sum over all multiset of request is not a problem when taking a slightly larger value of  $k$ . We can thus get similar bounds as long as  $d \frac{U_E}{D_E} \geq u'$  for some  $u' \geq \max\{1 + \frac{1}{c}, \mu\}$ .

Boxes with capacity lower than  $u'$  can be grouped with high upload capacity boxes to obtain the desired property as proposed in Lemma 1.

### Poor Upload Capacity Boxes

**Proof.**[of Lemma 1] Boxes  $b$  with upload capacity  $b < \mu$  are said to be *poor*. Boxes with upload capacity exactly  $\mu$  are said to be *medium*. Boxes  $b$  with  $u_b > \mu$  upload capacity are said to be *rich*. Let  $P$ ,  $M$  and  $R$  denote the sets of poor, medium and rich boxes respectively. We set  $n_P = |P|$ ,  $n_M = |M|$ , and  $n_R = |R|$  (Note that  $n = n_P + n_M + n_R$ ). Let  $u_P = \frac{U_P}{n_P}$  and  $u_R = \frac{U_R}{n_R}$  be the mean and overall upload capacities of poor and rich boxes respectively.

We construct  $B$  from  $A$  with same video allocation. For the sake of simplicity, we assume that  $s/\mu$  is an integral value as well as  $u_b s$  for each box  $b$ . In a pre-processing step, for each poor box  $b$ , we reserve  $\mu(1 - \frac{u_b}{\mu})s = \mu s - u_b s$  upload slots from the rich boxes. This upload will be used to forward  $(1 - \frac{u_b}{\mu})s$  stripes to the poor box and serve new arrivals in the swarm for up to  $(\mu - 1)(1 - \frac{u_b}{\mu})s$  stripes. This assignment should balance the overall number  $s_b$  of slots reserved on a rich box  $b$  such that its remaining upload capacity  $u'_b = u_b - \frac{s_b}{s}$  remains no less than  $\mu$ . (In a proportionally heterogeneous system, one would typically choose  $s_b$  proportional to  $u_b$ .) A corresponding space of  $\frac{s_b}{\mu s}$  should also be additionally be reserved for playback caching. We thus set  $d'_j = d_j + \frac{s_j}{\mu s}$ . This assignment is possible when  $U_R - \mu n_R \geq \mu n_P - U_P$ , i.e.  $u \geq \mu$ . Now we use a video allocation scheme for capacities  $u'_b, d'_b$  (where  $u'_b = u_b$  and  $d'_b = d_b$  for each poor or medium box  $b$ ). The connection scheduler works as previously except for the download connections of poor boxes. When a poor box  $b$  requests a video, the  $s$  stripes are downloaded from the boxes decided by the previous scheme. However,  $b$  downloads  $u_b \frac{s}{\mu}$  stripes directly but the  $(1 - \frac{u_b}{\mu})s$  others are downloaded via the rich boxes with reserved upload slots for box  $b$ . These rich boxes participate in the caching of the stripes they forward instead of  $b$ . This scheme allows to increase the overall upload capacity of the set  $E$  of all boxes caching some stripe requested by  $p$  boxes so that  $U_E \geq \mu p$ .  $\square$



---

Unité de recherche INRIA Rocquencourt  
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes  
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique  
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

---

Éditeur  
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399